ת"ב 2

שאלה 1

1. הפעולות שבחרנו לממש תומכות בכל הפעולות של BipartiteGraphTestDriver. בנוסף למתודות האלה, בחרנו להוסיף מתודה שמאפשרת מחיקה של קשת removeEdge, ומתודות נוספות שמספקות מידע על האיברים בגרף: graphContains, getNodeColor, nodesDifferent. הוספנוADT של צומת ואפשרות לגשת לצומת בעל תווית מסוימת (getNodeByLabel) על מנת שיהיה אפשר לבצע פעולות על הצמתים עצמם, כולל הפעולות של האובייקט שבתוך הצומת. פעולות אלה בשילוב עם הפעולות שנגזרו מה-BipartiteGraphTestDriver שימושיות לביצוע כל מיני אלגוריתמים על הגרף: לבנות אותו ע"י הוספת צמתים וקשתות, לערוך אותו ע"י מחיקת קשתות, להוציא מידע על הגרף ועל איבריו ולגשת לצמתים של הגרף. בין היתר, אוסף הפעולות שמימשנו מאפשר לעבור על כל הצמתים ולהפעיל איזושהי מתודה על הצמתים.
2. בחרנו לייצג את הגרף על ידי HashMap אשר ממפה מהתווית של הצומת אל הצומת. מחלקת הצומת מכיל שני ArrayLists: אחד של ילדיו ואחד של הוריו. מבני נתונים אלה מתוחזקים בכל פעולה על הגרף. משיקולי מודולריות ו-encapsulation בחרנו להגדיר מחלקה נפרדת עבור צומת. על מנת לאפשר גישה בזמן קבוע (O(1)) מהצומת לרשימת ילדיו והוריו, בחרנו להכיל את הרשימות של ילדים והורים בתוך הצומת ולתחזק אותן בכל הוספת צומת או קשת. חשוב לציין כי זמן הגישה מהתווית של הצומת אל ילדיו והוריו של הצומת הוא ברוב המקרים קבוע, אך במקרה הגרוע הוא O(log(n)), מאחר והגישה מהתווית אל הצומת מצריכה גישה לערך שב-HashMap ע"י מפתח. לא ניתן לספק גישה בזמן קבוע (במקרה הגרוע) מהתווית אל הצומת מבלי להחיל הגבלות על תווית (למשל אם היינו אומרים שהתווית היא מספר מ-1 עד-100, אז היה אפשר לספק גישה בזמן קבוע בעזרת LookUpTable).
3. הייצוג שבחרנו עבור הגרף נקרא adjacency list. לחילופין, היה ניתן לייצג את הגרף על ידי adjacency matrix. במסגרת הייצוג הזה, כל הצמתי ממוספרים מ-1 עד כמות הצמתים והחיבוריות בין הצמתים מיוצגת על ידי מטריצה של ערכי Boolean כשערך בכל תא מייצג האם קיימת קשת מהצומת שמספרו הוא מספר השורה של התא אל הצומת שמספרו הוא מספר העמודה של התא. ייצוג כזה בזבזני במקום, מסדר גודל של כמות הצמתים בריבוע, לכן בחרנו לא להשתמש בו.